2차시 과제 풀이

문제 목록

문제 번호	문제 이름	출처
BOJ 1766	문제집	
BOJ 14567	선수과목 (Prerequisite)	
BOJ 2748	피보나치 수 2	
BOJ 1463	1로 만들기	
BOJ 2293	동전 1	
BOJ 10164	격자상의 경로	KOI 2014 중등부
BOJ 3933	라그랑주의 네 제곱수 정리	
BOJ 9251	LCS	
BOJ 12865	평범한 배낭	
BOJ 5557	1학년	JOI 2011 예선
BOJ 17069	파이프 옮기기 2	
BOJ 2616	소형기관차	KOI 2003
BOJ 12869	뮤탈리스크	
BOJ 10836	여왕벌	KOI 2015
BOJ 1344	축구	
BOJ 11049	행렬 곱셈 순서	
BOJ 11062	카드 게임	2015 인터넷 예선
BOJ 10714	케이크 자르기 2	JOI 2015
BOJ 5550	헌책방	JOI 2011
BOJ 2518	회전 테이블	KOI 2012
BOJ 5463	건포도	IOI 2009 Day2
BOJ 5573	산책	JOI 2009
BOJ 2315	가로등 끄기	
BOJ 5466	상인	IOI 2009 Day2
BOJ 5813	이상적인 도시	IOI 2012 Day2
BOJ 17439	꽃집	2019 SNUPC Div.1

BOJ 1766 문제집

위상 정렬을 하는 문제인데, 가능한 위상 정렬 순서 중 사전 순으로 가장 작은 것을 구해야 한다.

`std::queue` 대신 `std::priority_queue`를 쓰면 된다.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, In[32323];
vector<int> G[32323];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1,s,e; i<=M; i++){
        cin >> s >> e;
        G[s].push_back(e);
        In[e]++;
    }

    priority_queue<int, vector<int>, greater<>> pq;
    for(int i=1; i<=N; i++) if(!In[i]) pq.push(i);
    while(pq.size()){
        int v = pq.top(); pq.pop();
        cout << v << " ";
        for(auto i : G[v]) if(!--In[i]) pq.push(i);
    }
}
```

BOJ 14567 선수과목 (Prerequisite)

$D_i := i$번 과목을 이수할 수 있는 최소 학기라고 정의하자.

i의 선수 과목 v에 대해 $D_v < D_i$가 성립해야 하므로 $D_i = \max(D_v) + 1$이다.

DAG 형태로 쉽게 표현할 수 있으므로 위상 정렬을 하면서 점화식을 계산하면 된다.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, In[1010], D[1010];
vector<int> G[1010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1,s,e; i<=M; i++){
        cin >> s >> e;
        G[s].push_back(e);
        In[e]++;
    }

    queue<int> q;
    for(int i=1; i<=N; i++) if(!In[i]) q.push(i), D[i] = 1;
    while(q.size()){
        int v = q.front(); q.pop();
```

```
        for(auto i : G[v]){
            if(!--In[i]) q.push(i);
            D[i] = max(D[i], D[v] + 1);
        }
    }
    for(int i=1; i<=N; i++) cout << D[i] << " ";
}
```

BOJ 2748 피보나치 수 2

$D_i := i$번째 피보나치 수

$D_i = D_{i-1} + D_{i-2}$

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

ll N, D[99] = {0, 1};
int main(){
    cin >> N;
    for(int i=2; i<=N; i++) D[i] = D[i-1] + D[i-2];
    cout << D[N];
}
```

BOJ 1463 1로 만들기

$D_i := i$에 연산을 적절히 취해서 1로 만드는 최소 연산 횟수

- 만약 $i > 1$이면 1을 빼는 연산을 수행할 수 있다 : $D_i = \min(D_i, D_{i-1} + 1)$
- 만약 $2|i$이면 2로 나누는 연산을 수행할 수 있다 : $D_i = \min(D_i, D_{i/2} + 1)$
- 만약 $3|i$이면 3으로 나누는 연산을 수행할 수 있다 : $D_i = \min(D_i, D_{i/3} + 1)$

```
#include <bits/stdc++.h>
using namespace std;

int N, D[1010101];
int main(){
    cin >> N;
    for(int i=2; i<=N; i++){
        D[i] = D[i-1] + 1;
        if(i % 2 == 0) D[i] = min(D[i], D[i/2] + 1);
        if(i % 3 == 0) D[i] = min(D[i], D[i/3] + 1);
    }
    cout << D[N];
}
```

BOJ 2293 동전 1

$D_i := i$원을 지불하는 방법의 수

- 0원을 지불하는 방법은 1가지(공집합도 집합임) : $D_0 = 1$
- 마지막에 x원짜리 동전을 사용해서 i원을 지불한 경우 : $D_i = D_i + D_{i-x}$

```
#include <bits/stdc++.h>
```

```cpp
using namespace std;
using ll = long long;

int N, K, A[111], D[10101];
int main(){
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> A[i];

    D[0] = 1;
    for(int c=1; c<=N; c++){ // 1...c번째 동전만 사용해서 i원을 만드는 경우
        for(int j=1; j<=K; j++){
            if(j-A[c] >= 0) D[j] += D[j-A[c]];
        }
    }
    cout << D[K];
}
```

BOJ 10164 격자상의 경로

$(1,1)$에서 (y,x)를 거쳐서 (N,M)으로 가는 경우의 수는 ($(1,1)$에서 (y,x)로 가는 경우의 수)와 ((y,x)에서 (N,M)으로 가는 경우의 수)를 곱한 것과 동일하다. 그러므로 $D[i][j]$를 아래와 같이 정의하면, 문제의 정답은 $D[y][x] \cdot D[N-y+1][M-x+1]$이 된다.

$D[i][j] :=$ $(1,1)$에서 (i,j)로 이동하는 경우의 수 (아래로 $i-1$칸, 오른쪽으로 $j-1$칸 이동하는 경우의 수)

- $(1,1)$에 도달하는 방법은 1가지 : $D[1][1] = 1$
- $(i-1,j)$에서 아래로 한 칸 이동해 (i,j)에 도달할 수 있다 : $D[i][j] = D[i][j] + D[i-1][j]$
- $(i,j-1)$에서 위로 한 칸 이동해 (i,j)에 도달할 수 있다 : $D[i][j] = D[i][j] + D[i][j-1]$

반드시 지나야 하는 칸 (y,x)가 없다면 $(y,x) = (N,M)$으로 생각하면 된다.

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, M, K, D[16][16];
int main(){
    cin >> N >> M >> K;
    if(K == 0) K = N * M;
    int y = (K-1) / M + 1, x = (K-1) % M + 1;

    D[1][1] = 1;
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            if(i == 1 && j == 1) continue;
            D[i][j] = D[i-1][j] + D[i][j-1];
        }
    }

    cout << D[y][x] * D[N-y+1][M-x+1];
}
```

$D[i][j] = \frac{(i+j-2)!}{(i-1)!(j-1)!}$을 이용하는 방법도 있는데, $28! \approx 3 \cdot 10^{29}$라서 계산 중간 과정에서 `__int128_t`를 사용해야 한다.

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, M, K;
__int128_t Fac[30];

ll D(int i, int j){
    return Fac[i+j-2] / Fac[i-1] / Fac[j-1];
}

int main(){
    cin >> N >> M >> K;
    if(K == 0) K = N * M;
    int y = (K-1) / M + 1, x = (K-1) % M + 1;

    Fac[0] = 1;
    for(int i=1; i<=28; i++) Fac[i] = Fac[i-1] * i;

    cout << D(y, x) * D(N-y+1, M-x+1);
}
```

BOJ 3933 라그랑주의 네 제곱수 정리

$D(i, j) :=$ 제곱 수 j개를 써서 i를 만드는 경우의 수

점화식을 채우는 순서에 주의해야 함

```cpp
#include <bits/stdc++.h>
using namespace std;
constexpr int SZ = 1 << 15;

int D[SZ][5];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    for(int i=1; i*i<SZ; i++){
        D[i*i][1] = 1;
        for(int j=1; j+i*i<SZ; j++){
            D[j+i*i][2] += D[j][1];
            D[j+i*i][3] += D[j][2];
            D[j+i*i][4] += D[j][3];
        }
    }

    while(true){
        int N; cin >> N; if(!N) break;
        cout << D[N][1] + D[N][2] + D[N][3] + D[N][4] << "\n";
    }
}
```

BOJ 9251 LCS

$D(i,j) := A_1A_2\cdots A_i$와 $B_1B_2\cdots B_j$의 LCS의 길이

- $i=0$이거나 $j=0$이면 $D(i,j)=0$
- $A_i=B_j$이면 A_i와 B_j를 매칭할 수 있으므로 $D(i,j)=D(i-1,j-1)+1$
- $A_i\neq B_j$이면 $D(i,j)=\max(D(i-1,j),D(i,j-1))$

```
#include <bits/stdc++.h>
using namespace std;

int N, M, D[1010][1010];
string A, B;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> A >> B;
    N = A.size(); M = B.size();

    A = "#" + A; B = "@" + B; // change to 1-based
    for(int i=1; i<=N; i++){
        for(int j=1; j<=M; j++){
            if(A[i] == B[j]) D[i][j] = D[i-1][j-1] + 1;
            else D[i][j] = max(D[i-1][j], D[i][j-1]);
        }
    }
    cout << D[N][M];
}
```

BOJ 12865 평범한 배낭

$D(i,j) := 1\cdots i$번째 물건으로 j만큼의 무게를 채울 때 가치의 최댓값

- i번째 물건을 사용하지 않으면 $D(i,j)\leftarrow D(i-1,j)$
- i번째 물건을 사용하면 $D(i,j)\leftarrow D(i-1,j-W_i)+V_i$

```
#include <bits/stdc++.h>
using namespace std;

int N, K, W[111], V[111];
int D[111][101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> W[i] >> V[i];
    for(int i=1; i<=N; i++){
        for(int j=1; j<=K; j++){
            D[i][j] = D[i-1][j];
            if(j-W[i] >= 0) D[i][j] = max(D[i][j], D[i-1][j-W[i]]+V[i]);
        }
    }
    cout << *max_element(D[N], D[N]+K+1);
}
```

DP 배열을 일차원으로 잡을 수도 있다. 시간 복잡도는 동일하지만, 캐시 적중률이 높아져서 더 빠르다.

$D(i)$:= 무게를 i만큼 채울 때 가치의 최댓값

j에 대한 for문 순서에 유의하자. 0부터 K까지 돌리면 한 물건을 여러 개 가져갈 수 있다.

```
#include <bits/stdc++.h>
using namespace std;

int N, K, W[111], V[111];
int D[101010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> W[i] >> V[i];
    for(int i=1; i<=N; i++){
        for(int j=K; j>=W[i]; j--){
            D[j] = max(D[j], D[j-W[i]]+V[i]);
        }
    }
    cout << *max_element(D, D+K+1);
}
```

BOJ 5557 1학년

$D(i, j)$:= i번째 수까지 이용해서 j를 만드는 경우의 수

- $D(1, A_1) = 1$
- A_i 앞에 `+`를 붙이는 경우 : $D(i, j) \leftarrow D(i, j) + D(i, j - A_i)$
- A_i 앞에 `-`를 붙이는 경우 : $D(i, j) \leftarrow D(i, j) + D(i, j + A_i)$

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, A[111];
ll D[111][22];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];

    D[1][A[1]] = 1;
    for(int i=2; i<N; i++){
        for(int j=0; j<=20; j++){
            if(j-A[i] >= 0) D[i][j] += D[i-1][j-A[i]];
            if(j+A[i] <= 20) D[i][j] += D[i-1][j+A[i]];
        }
    }
    cout << D[N-1][A[N]];
}
```

BOJ 17069 파이프 옮기기 2

- $D(i,j,0) :=$ 파이프가 가로 방향으로 놓여있을 때 오른쪽 끝점이 (i,j)인 경우의 수
- $D(i,j,1) :=$ 파이프가 세로 방향으로 놓여있을 때 아래쪽 끝점이 (i,j)인 경우의 수
- $D(i,j,2) :=$ 파이프가 대각선 방향으로 놓여있을 때 오른쪽 아래 끝점이 (i,j)인 경우의 수

```
#include <bits/stdcd++.h>
using namespace std;
using ll = long long;

int N, A[55][55];
ll D[55][55][3]; // - | \

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) for(int j=1; j<=N; j++) cin >> A[i][j];

    D[1][2][0] = 1;
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++){
            if(A[i][j]) continue;
            D[i][j][0] += D[i][j-1][0] + D[i][j-1][2];
            D[i][j][1] += D[i-1][j][1] + D[i-1][j][2];
            if(!A[i-1][j] && !A[i][j-1]){
                D[i][j][2] += D[i-1][j-1][0] + D[i-1][j-1][1] + D[i-1][j-1][2];
            }
        }
    }
    cout << D[N][N][0] + D[N][N][1] + D[N][N][2];
}
```

BOJ 2616 소형기관차

가장 먼저 해야하는 관찰은, 기관차는 항상 K개의 객차를 가져가는 것이 최적이라는 것이다.

$D(i,j) :=$ i개의 기관차를 이용해 $1 \cdots j$번 객차를 적절히 운송할 때, 운송할 수 있는 손님 수의 최댓값

- i번째 기관차가 $j-K+1, j-K+2, \cdots, j$번 객차를 운송하면 :
  $D(i,j) \leftarrow D(i-1, j-K) + \sum_{k=0}^{K} A_{j-k}$
- 그렇지 않는다면 : $D(i,j) \leftarrow D(i, j-1)$

A_i의 부분합을 구하는 것은 누적합을 이용하면 됨

```
#include <bits/stdc++.h>
using namespace std;

int N, K, A[50505], D[4][50505], mx;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];
    cin >> K;
    partial_sum(A+1, A+N+1, A+1);
    for(int i=1; i<=3; i++){
        for(int j=i*K; j<=N; j++){
```

```
            D[i][j] = max(D[i][j-1], D[i-1][j-K] + A[j] - A[j-K]);
            mx = max(mx, D[i][j]);
        }
    }
    cout << mx;
}
```

BOJ 12869 뮤탈리스크

$D(a,b,c) :=$ SCV의 체력이 각각 a,b,c일 때, 모든 SCV를 잡기 위해서 뮤탈리스크가 공격해야 하는 최소 횟수

$a \le b \le c$로 관리하는 것이 정신 건강에 이롭다.

```
#include <bits/stdc++.h>
using namespace std;

int N, A[3], D[66][66][66];

int f(int a, int b, int c){
    int &res = D[a][b][c];
    if(res != -1) return res;
    if(a == 0 && b == 0 && c == 0) return res = 0;

    int damage[3] = {1, 3, 9}, nxt[3];
    res = 0x3f3f3f3f;
    do{
        nxt[0] = max(0, a - damage[0]);
        nxt[1] = max(0, b - damage[1]);
        nxt[2] = max(0, c - damage[2]);
        sort(nxt, nxt+3);
        res = min(res, f(nxt[0], nxt[1], nxt[2]) + 1);
    }while(next_permutation(damage, damage+3));
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=0; i<N; i++) cin >> A[i];
    sort(A, A+3);
    memset(D, -1, sizeof D);
    cout << f(A[0], A[1], A[2]);
}
```

BOJ 10836 여왕벌

첫 번째 행과 첫 번째 열의 최종 값만 알고 있다면, 나머지 부분은 DP로 채워줄 수 있다.

- $D(i,j) = \max(D(i-1,j), D(i,j-1))$

첫 행/열을 Naive하게 채우면 $O(NM)$이고 (아마도) TLE를 받게 된다.

배열 $\text{border}[] = \{(M,1),(M-1,1),\cdots,(2,1)(1,1),(1,2),\cdots(1,N)\}$을 생각해보자. 각 날마다 아래와 같은 연산을 수행해야 한다.

- $\text{border}[0\cdots(a-1)]$에 0 더함

- $\text{border}[a \cdots (a+b-1)]$에 1 더함
- $\text{border}[(a+b) \cdots]$에 2 더함

Prefix Sum을 이용하면 $O(N+M)$에 수행할 수 있다.

- 모든 쿼리에 대해, `sum[a]++; sum[a+b]++;`
- 마지막에 `for(int i=1; ; i++) sum[i] += sum[i-1];`

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;

int N, Q, Sum[1414], D[777][777];
vector<PII> Border;

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> Q;
    for(int i=N; i>=1; i--) Border.emplace_back(i, 1);
    for(int j=2; j<=N; j++) Border.emplace_back(1, j);
    for(int i=1; i<=Q; i++){
        int a, b, c; cin >> a >> b >> c;
        Sum[a]++; Sum[a+b]++;
    }
    partial_sum(Sum, Sum+1414, Sum);
    for(int i=0; i<Border.size(); i++){
        D[Border[i].x][Border[i].y] += Sum[i];
    }
    for(int i=2; i<=N; i++){
        for(int j=2; j<=N; j++){
            D[i][j] = max({D[i-1][j], D[i][j-1], D[i-1][j-1]});
        }
    }
    for(int i=1; i<=N; i++){
        for(int j=1; j<=N; j++) cout << D[i][j] + 1 << " ";
        cout << "\n";
    }
}
```

BOJ 1344 축구

$D(i,j,k) := i$번째 간격까지 A팀이 j점, B팀이 k점 득점할 확률

- A팀만 득점할 확률 : $A \cdot (1-B)$
- B팀만 득점할 확률 : $(1-A) \cdot B$
- 두 팀 모두 득점할 확률 : $A \cdot B$
- 두 팀 모두 득점하지 못할 확률 : $(1-A) \cdot (1-B)$

```
#include <bits/stdc++.h>
using namespace std;

constexpr int N = 18;
double A, B, D[22][22][22];

bool isPrime(int n){
```

```
    if(n < 2) return false;
    for(int i=2; i*i<=n; i++) if(n % i == 0) return false;
    return true;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> A >> B; A /= 100; B /= 100;
    D[0][0][0] = 1;
    for(int i=1; i<=N; i++){
        for(int j=0; j<=i; j++){
            for(int k=0; k<=i; k++){
                if(j > 0) D[i][j][k] += D[i-1][j-1][k] * A * (1-B);
                if(k > 0) D[i][j][k] += D[i-1][j][k-1] * (1-A) * B;
                if(j > 0 && k > 0) D[i][j][k] += D[i-1][j-1][k-1] * A * B;
                D[i][j][k] += D[i-1][j][k] * (1-A) * (1-B);
            }
        }
    }

    double ans = 0;
    for(int i=0; i<=N; i++) for(int j=0; j<=N; j++)
        if(isPrime(i) || isPrime(j)) ans += D[18][i][j];

    cout << fixed << setprecision(10) << ans;
}
```

BOJ 11049 행렬 곱셈 순서

$D(i,j) := i$번째 행렬부터 j번째 행렬까지 곱하는 최소 비용

- $D(i,i+1) = R_iC_iC_{i+1}$
- $D(i,j) = \min(D(i,k) + D(k+1,j) + R_iC_kC_j)$
  - $D(i,k) : i \cdots k$번째 행렬($R_i \times C_k$)을 곱하는 최소 비용
  - $D(k+1,j) : k+1 \cdots j$번째 행렬($R_{k+1} \times C_j$)을 곱하는 최소 비용
  - $R_iC_kC_j$: 두 행렬을 곱하는 비용

$j-i$가 작은 것부터 채워야 함

```
#include <bits/stdc++.h>
using namespace std;

int N, R[555], C[555], D[555][555];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=1; i<=N; i++) cin >> R[i] >> C[i];

    memset(D, 0x3f, sizeof D);
    for(int i=1; i<=N; i++) D[i][i] = 0;
    for(int i=1; i<N; i++) D[i][i+1] = R[i] * C[i] * C[i+1];

    for(int t=2; t<=N; t++){
        for(int i=1; i+t<=N; i++){
            int j = i + t;
```

```
            for(int k=i; k<j; k++) D[i][j] = min(D[i][j], D[i][k] + D[k+1][j] +
R[i]*C[k]*C[j]);
        }
    }
    cout << D[1][N];
}
```

BOJ 11062 카드 게임

근우의 차례에서는 근우의 점수가 최대가 되도록, 명우의 차례에서는 근우의 점수가 최소가 되도록 플레이한다. (minimax tree 느낌으로)

$D(i, j, turn) :=$ 왼쪽 끝에 i, 오른쪽 끝에 j번 카드가 있는 상황에서 근우가 얻을 수 있는 최대 점수. 단, $turn$이 0이면 근우, 1이면 명우의 차례

- $D(i, j, 0) = \max(D(i+1, j, 1) + A_i, D(i, j-1, 1) + A_j)$
- $D(i, j, 1) = \min(D(i+1, j, 0), D(i, j-1, 0))$

```
#include <bits/stdc++.h>
using namespace std;

int N, A[1010], D[1010][1010][2];

int f(int i, int j, int turn){
    int &res = D[i][j][turn];
    if(res != -1) return res;
    if(i >= j) return res = (turn ? 0 : A[i]);

    if(turn == 0) res = max(f(i+1, j, 1) + A[i], f(i, j-1, 1) + A[j]);
    else res = min(f(i+1, j, 0), f(i, j-1, 0));
    return res;
}

void Solve(){
    cin >> N;
    for(int i=1; i<=N; i++) cin >> A[i];

    memset(D, -1, sizeof D);
    cout << f(1, N, 0) << "\n";
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    int T; cin >> T;
    while(T--) Solve();
}
```

BOJ 10714 케이크 자르기 2

원형이 아닌 선형이라고 생각해보자.

$D(l, r, turn) := l \cdots r$번 조각을 가져간 상황에서, 앞으로 JOI가 가져올 수 있는 조각의 최댓값이라고 정의하자. 단, $turn$이 0이면 IOI의 차례이고, 1이면 JOI의 차례이다.

- $turn = 0$: IOI는 A_{l-1}과 A_{r+1} 중 더 큰 것을 가져간다.
  - $A_{l-1} > A_{r+1}$이면 $D(l, r, 0) = D(l-1, r, 1)$

    - $A_{l-1} \le A_{r+1}$이면 $D(l,r,0) = D(l,r+1,1)$
- $turn$: 1 : JOI는 A_{l-1} 혹은 A_{r+1}을 가져갔을 때 최종적으로 더 큰 결과를 갖게 되는 쪽으로 선택한다.
    - $D(l,r,1) = \max(D(l-1,r,0) + A_{l-1}, D(l,r+1,0) + A_{r+1})$

IOI의 차례는 굳이 메모이제이션할 필요 없다.

이 문제는 선형이 아니라 원형이므로 $l-1, r+1$ 대신 $(l-1) \mod N, (r+1) \mod N$을 사용하면 된다.

```
#include <bits/stdc++.h>
using namespace std;
using ll = long long;

int N, A[2020];
ll D[2020][2020];

int next(int v){ return (v + 1) % N; }
int prev(int v){ return (v + N - 1) % N; }

ll f(int l, int r, int turn){
    if(turn == 0){ // IOI
        if(next(r) == l) return 0;
        if(A[prev(l)] > A[next(r)]) return f(prev(l), r, 1);
        else return f(l, next(r), 1);
    }

    ll &res = D[l][r];
    if(res != -1) return res;
    if(next(r) == l) return res = 0;

    ll t1 = f(prev(l), r, 0) + A[prev(l)];
    ll t2 = f(l, next(r), 0) + A[next(r)];
    return res = max(t1, t2);
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=0; i<N; i++) cin >> A[i];

    ll ans = 0;
    memset(D, -1, sizeof D);
    for(int i=0; i<N; i++) ans = max(ans, f(i, i, 0) + A[i]);
    cout <<ans;
}
```

BOJ 5550 헌책방

동일한 장르의 책을 매입하는 경우에는 가격이 비싼 것부터 매입하는 것이 무조건 이득이다.

$C(i,j) := i$번째 장르의 책을 j권 매입할 때의 최대 가격

$D(i,j) := 1 \cdots i$번째 장르의 책을 총 j권 매입할 때의 최대 가격

C는 각 장르 별로 책을 가격에 대한 내림차순으로 정렬하고 Prefix Sum을 구하면 된다. D는 DP로 구할 수 있다.

```cpp
#include <bits/stdc++.h>
using namespace std;

int N, K, A[11][2020], ptr[11];
int C[11][2020], D[11][2020];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++){
        int c, g; cin >> c >> g;
        A[g][++ptr[g]] = c;
    }
    for(int i=1; i<=10; i++){
        sort(A[i]+1, A[i]+ptr[i]+1, greater<>());
        partial_sum(A[i]+1, A[i]+ptr[i]+1, C[i]+1);
        for(int j=1; j<=ptr[i]; j++) C[i][j] += j * (j - 1);
    }
    for(int i=1; i<=10; i++){
        for(int j=1; j<=N; j++){
            for(int k=0; k<=j; k++) D[i][j] = max(D[i][j], D[i-1][k] + C[i][j-
k]);
        }
    }
    cout << D[10][K];
}
```

BOJ 2518 회전 테이블

$D(a, b, c, last)$:= 첫 번째 사람이 a개, 두 번째 사람이 b개, 세 번째 사람이 c개를 먹었고, 마지막으로 음식을 먹은 사람이 $last$인 상황에서, 세 명의 사람이 원하는 만큼 음식을 먹을 수 있도록 만드는 최소 회전 횟수

$O(P_1P_2P_3)$에 문제를 해결할 수 있다.

```cpp
#include <bits/stdc++.h>
using namespace std;
constexpr int INF = 0x3f3f3f3f;

int N, V[3][111], P[3], D[111][111][111][3];

int rotate(int a, int b){
    int d = abs(a - b) % N;
    return min(d, N-d);
}

int f(int a, int b, int c, int last){
    int &res = D[a][b][c][last];
    if(res != -1) return res;
    if(a == P[0] && b == P[1] && c == P[2]) return res = 0;
    if(a  > P[0] || b  > P[1] || c  > P[2]) return res = INF;

    res = INF;
    int now = array<int,3>{V[0][a], V[1][b]+N/3*2, V[2][c]+N/3}[last];
    res = min(res, f(a+1, b, c, 0) + rotate(now, V[0][a+1]));
    res = min(res, f(a, b+1, c, 1) + rotate(now, V[1][b+1]+N/3*2));
```

```
    res = min(res, f(a, b, c+1, 2) + rotate(now, V[2][c+1]+N/3));
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    for(int i=0; i<3; i++){
        cin >> P[i];
        for(int j=1; j<=P[i]; j++) cin >> V[i][j], V[i][j]--;
    }
    V[1][0] = N/3;
    V[2][0] = N/3*2;
    memset(D, -1, sizeof D);
    cout << f(0, 0, 0, 0);
}
```

BOJ 5463 건포도

$D(r_1, c_1, r_2, c_2) :=$ 꼭짓점이 $(r_1, c_1), (r_2, c_2)$인 초콜릿을 자를 때 지불해야 하는 건포도의 최솟값이라고 정의하자. 상태 전이는 가로 방향으로 분할하는 $r_2 - r_1$가지와 세로 방향으로 분할하는 $c_2 - c_1$가지로, 총 $(r_2 - c_1) + (c_2 - c_1)$가지가 존재한다.

직사각형 영역에 속한 건포도의 개수는 2D Prefix Sum을 이용해 $O(1)$에 구할 수 있으므로 각 상태 전이는 $O(1)$ 시간에 처리할 수 있다.

상태 공간의 크기가 $O(N^2M^2)$이고, 각 상태의 답을 구하는데 $O(N+M)$이 걸리므로 전체 시간 복잡도는 $O(N^2M^2(N+M)) = O(\max(N, M)^5)$이다.

```
#include <bits/stdc++.h>
using namespace std;

int N, M, A[55][55], D[55][55][55][55];

int f(int r1, int c1, int r2, int c2){
    int &res = D[r1][c1][r2][c2];
    if(res != -1) return res;
    if(r1 == r2 && c1 == c2) return res = 0;

    res = 0x3f3f3f3f;
    for(int i=r1; i<r2; i++) res = min(res, f(r1, c1, i, c2) + f(i+1, c1, r2,
c2));
    for(int j=c1; j<c2; j++) res = min(res, f(r1, c1, r2, j) + f(r1, j+1, r2,
c2));
    res += A[r2][c2] - A[r2][c1-1] - A[r1-1][c2] + A[r1-1][c1-1];
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) cin >> A[i][j];
    for(int i=1; i<=N; i++) for(int j=1; j<=M; j++) A[i][j] += A[i-1][j] + A[i]
[j-1] - A[i-1][j-1];
    memset(D, -1, sizeof D);
    cout << f(1, 1, N, M);
}
```

BOJ 5573 산책

어떤 지점을 홀수 번 방문하면 방향이 반대로 바뀌게 된다. 처음 $N-1$번의 산책에서 각 지점을 몇 번 방문하는지 계산하면 N번째 산책의 경로를 구할 수 있다.

(i,j)를 K번 방문했다고 하자. 만약 K가 짝수라면 $(i+1,j)$와 $(i,j+1)$을 각각 $K/2$번 씩 방문하게 된다. 만약 K가 홀수라면 한 곳은 $(K+1)/2$번, 다른 곳은 $(K-1)/2$번 방문하게 된다.

(i,j)에서 $(i+1,j)$과 $(i,j+1)$로 값을 전파하기 때문에 DP로 계산할 수 있다.

```
#include <bits/stdc++.h>
using namespace std;
constexpr int di[] = {1, 0}; // down, right
constexpr int dj[] = {0, 1}; // down, right

int H, W, N, A[1010][1010], D[1010][1010];

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> H >> W >> N;
    for(int i=1; i<=H; i++) for(int j=1; j<=W; j++) cin >> A[i][j];
    D[1][1] = N - 1;
    for(int i=1; i<=H; i++){
        for(int j=1; j<=W; j++){
            int flag = A[i][j];
            D[i+di[flag]][j+dj[flag]] += (D[i][j] + 1) / 2;
            D[i+di[flag^1]][j+dj[flag^1]] += D[i][j] / 2;
            if(D[i][j] & 1) A[i][j] ^= 1;
        }
    }
    int i = 1, j = 1;
    while(i <= H && j <= W){
        i += di[A[i][j]];
        j += dj[A[i][j]];
    }
    cout << i << " " << j;
}
```

BOJ 2315 가로등 끄기

몇 가지 관찰을 하자.

1. 가로등을 끄는데 시간이 걸리지 않기 때문에, 어떤 가로등을 지나간다면 그 가로등을 끄는 것이 이득임
2. (1)에 의해, i번째 가로등과 j번째 가로등이 꺼져있다면, $x \in [i,j]$번째 가로등은 모두 꺼져있음
3. (1)에 의해, i번째 가로등과 j번째 가로등이 켜져있다면, $x \in [i,j]$번째 가로등은 모두 켜져있음

$x \in [i,j]$번째 가로등만 꺼져있는 상태에 대한 DP를 생각해볼 수 있다. 구체적으로, 꺼져있는 가로등의 구간 $[i,j]$와 로봇의 현재 위치 p에 대한 DP를 하면 된다.

$x \in [i,j]$번째 가로등만 꺼져있다면 로봇은 무조건 i 혹은 j에 있기 때문에, 각 구간마다 로봇의 현재 위치 p로 가능한 지점은 2가지이다. 그러므로 아래와 같은 DP를 생각할 수 있다.

- $D(i,j,flag) :=$ $x \in [i,j]$번째 가로등만 꺼져있을 때, 남은 가로등을 모두 끌 때까지 낭비되는 전력의 최솟값. 단, $flag$가 0이면 로봇은 i번째 가로등에 있고, 1이면 j번째 가로등에 있다.
- 다음 상태로 가능한 것은 로봇이 $i-1$과 $j+1$로 가는 경우 밖에 없으므로, $O(N^2)$에 점화식을 계산할 수 있다.

```cpp
#include <bits/stdc++.h>
using namespace std;

using ll = long long;

int N, M;
ll D[1010][1010][2];
ll Pos[1010], Pow[1010], Sum[1010];

ll range_sum(int l, int r){
    return Sum[r] - Sum[l-1];
}

ll f(int l, int r, int flag){
    ll &res = D[l][r][flag];
    if(res != -1) return res;
    if(l == 1 && r == N) return res = 0;
    res = 0x3f3f3f3f3f3f3f3f;

    int now = flag ? r : l;
    ll power_on = range_sum(1, l-1) + range_sum(r+1, N);
    if(l > 1) res = min(res, f(l-1, r, 0) + (Pos[now] - Pos[l-1]) * power_on);
    if(r < N) res = min(res, f(l, r+1, 1) + (Pos[r+1] - Pos[now]) * power_on);
    return res;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> M;
    for(int i=1; i<=N; i++) cin >> Pos[i] >> Pow[i];
    partial_sum(Pow+1, Pow+N+1, Sum+1);

    memset(D, -1, sizeof D);
    cout << f(M, M, 0);
}
```

BOJ 5466 상인

(T_i, L_i, M_i)가 $(0, S, 0), (505050, S, 0)$인 시장을 추가해서 시작점과 끝점을 표현하자.

Subtask 1. $N \leq 5\,000$, T_i**는 모두 다름**

시장을 T_i 기준으로 정렬하면 $O(N^2)$ DP 풀이는 쉽게 떠올릴 수 있다.

- $D_i :=$ 마지막으로 방문한 시장이 i번째 시장일 때의 최대 이득
- $D_i = \max_{0 \leq j < i}(D_j + M_i - \mathrm{cost}(L_j, L_i))$
  - $\mathrm{cost}(s, e) := s$에서 e로 가는 비용
- 시작점을 0번째 시장, 끝점을 $N+1$번째 시장이라고 하면 D_{N+1}이 정답이 된다.

코드 : https://oj.uz/submission/432936

Subtask 2. $N \leq 500\,000$, T_i는 모두 다름

$O(N \log^k N)$ 정도에 동작하는 풀이를 찾아야 한다. 점화식을 잘 정리해보자.

- $L_j < L_i$인 경우: $D_i = \max(D_j + L_j D) - L_i D + M_i$
- $L_j > L_i$인 경우: $D_i = \max(D_j - L_j U) + L_i U + M_i$

max 안에 있는 식$(D_j + L_j D, D_j - L_j U)$의 최댓값은 구간의 최댓값을 구하는 연산이므로 세그먼트 트리를 이용해 $O(\log N)$에 구할 수 있다. 각 i에 대해 D_i를 $O(\log N)$ 시간에 계산할 수 있으므로 전체 시간 복잡도는 $O(N \log N)$이 된다.

코드 : https://oj.uz/submission/432956

Subtask 3. $N \leq 500\,000$

같은 날에 여러 개의 시장이 열릴 수 있고, 여러 개의 시장이 동시에 열리면 모두 갈 수 있다. 중요한 관찰이 필요하다.

- 시장을 방문하는 시간은 계산하지 않기 때문에, 이동하면서 만나는 시장은 모두 방문하는 것이 이득이다. 그러므로 같은 날에 방문하는 시장은 연속된 구간을 이루게 된다.

각 날짜 별로 순서대로 답을 구할 것이다. 시장을 각 날짜 별로 구분해서 저장하고(`std::vector<Info> v[505050]`), 같은 날짜 안에서는 시장의 위치 순서대로 정렬하자.

같은 날에 방문하는 시장은 연속한 구간 형태이므로, 현재 위치 pos에 도달했을 때 왼쪽으로 가는 중인지 오른쪽으로 가는 중인지 알고 있어야 한다. 배열 2개를 추가로 만들자.

- $DL_i :=$ 상인이 해당 날짜에 i번째 시장에 도착했을 때의 최대 이득 (단, 상인은 왼쪽으로 이동하는 중)
- $DR_i :=$ 상인이 해당 날짜에 i번째 시장에 도착했을 때의 최대 이득 (단, 상인은 오른쪽으로 이동하는 중)
- 해당 날짜에서 마지막에 i번째 시장을 방문한 경우의 최대 이득은 $\max(DL_i, DR_i)$

DL_i, DR_i의 초깃값은 Subtask 2처럼 세그먼트 트리를 이용해 전날 방문한 시장에서 가져오면 된다. 초기화를 한 다음에는 아래 수식을 이용해 값을 구할 수 있다.

- $i + 1 <$ size이면 $DL_i = \min(DL_i, DL_{i+1} - U(L_{i+1} - L_i) + M_i)$
- $i > 0$이면 $DR_i = \min(DR_i, DR_{i-1} - D(L_i - L_{i-1}) + M_i)$

나머지 부분은 Subtask 2와 비슷하게 구현하면 된다.

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using PII = pair<int, int>;

constexpr int SZ = 1 << 19;
struct SegmentTree{
    int T[SZ << 1];
    SegmentTree(){ memset(T, 0xc0, sizeof T); }
    void update(int x, int v){
        x |= SZ; T[x] = max(T[x], v);
        while(x >>= 1) T[x] = max(T[x], v);
    }
    int query(int l, int r){
        l |= SZ; r |= SZ; int ret = T[0];
        while(l <= r){
            if(l & 1) ret = max(ret, T[l++]);
```

```
            if(~r & 1) ret = max(ret, T[r--]);
            l >>= 1; r >>= 1;
        }
        return ret;
    }
};

int N, U, D, S, DP[505050];
vector<PII> A[505050];

SegmentTree T_D, T_U;

void Update(int pos, int dp){
    T_D.update(pos, dp + D * pos);
    T_U.update(pos, dp - U * pos);
}

int Query(int pos, int money){
    int dw = T_D.query(0, pos) - D * pos + money;
    int up = T_U.query(pos, 505050) + U * pos + money;
    return max(dw, up);
}

void SolveTime(int ti){
    if(A[ti].empty()) return;
    int len = A[ti].size();

    vector<int> DL(len), DR(len);
    for(int i=0; i<len; i++){
        DL[i] = DR[i] = Query(A[ti][i].x, A[ti][i].y);
    }
    for(int i=len-2; i>=0; i--){
        DL[i] = max(DL[i], DL[i+1] - U * (A[ti][i+1].x - A[ti][i].x) + A[ti]
[i].y);
    }
    for(int i=1; i<len; i++){
        DR[i] = max(DR[i], DR[i-1] - D * (A[ti][i].x - A[ti][i-1].x) + A[ti]
[i].y);
    }
    for(int i=0; i<len; i++){
        int res = max(DL[i], DR[i]);
        Update(A[ti][i].x, res);
    }
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> U >> D >> S;
    int sz = 0;
    for(int i=1; i<=N; i++){
        int t, l, m; cin >> t >> l >> m;
        A[t].emplace_back(l, m);
        sz = max(sz, t);
    }
    A[0].emplace_back(S, 0);    // start
    A[++sz].emplace_back(S, 0); // end
    for(int i=0; i<=sz; i++) sort(A[i].begin(), A[i].end());
```

```
    memset(DP, 0xc0, sizeof DP);
    DP[0] = 0; Update(S, 0);
    for(int i=1; i<=sz; i++) SolveTime(i);
    cout << Query(S, 0);
}
```

BOJ 5813 이상적인 도시

https://justicehui.github.io/ioi/2019/03/09/BOJ5813/

```
#include <bits/stdc++.h>
#define x first
#define y second
#define all(v) v.begin(), v.end()
#define compress(v) sort(all(v)), v.erase(unique(all(v)), v.end())
using namespace std;

using ll = long long;
using PII = pair<int, int>;
constexpr int MOD = 1e9;
constexpr int di[] = {1, -1, 0, 0};
constexpr int dj[] = {0, 0, 1, -1};

int N;
PII A[101010];
int S[101010];
vector<int> G[101010];

int getSize(int v, int b=-1){
    for(auto i : G[v]) if(i != b) S[v] += getSize(i, v);
    return S[v];
}

ll Solve(){
    memset(S, 0, sizeof S);
    for(int i=1; i<=N; i++) G[i].clear();
    sort(A+1, A+N+1);

    int pv = 0;
    map<PII, int> mp;
    for(int i=1; i<=N; i++){
        if(i != 1 && A[i-1].x == A[i].x && A[i-1].y+1 == A[i].y) mp[A[i]] = pv;
        else mp[A[i]] = ++pv;
        S[pv]++;
    }

    for(int i=1; i<=N; i++){
        int now = mp[A[i]];
        for(int k=0; k<4; k++){
            int nxt = mp[{A[i].x+di[k], A[i].y+dj[k]}];
            if(nxt && now != nxt) G[now].push_back(nxt);
        }
    }
    for(int i=1; i<=pv; i++) compress(G[i]);

    getSize(1);
    ll ret = 0;
```

```
    for(int i=1; i<=N; i++) ret += 1LL * S[i] * (N - S[i]) % MOD;
    return ret % MOD;
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N;
    ll ans = 0;
    for(int i=1; i<=N; i++) cin >> A[i].x >> A[i].y;
    ans += Solve();
    for(int i=1; i<=N; i++) swap(A[i].x, A[i].y);
    ans += Solve();
    if(ans >= MOD) ans -= MOD;
    cout << ans;
}
```

BOJ 17439 꽃집

https://justicehui.github.io/ps/2020/04/21/BOJ17439/

개인적으로 Monotone Queue Optimization 짜는 것보다 Li-Chao Tree 쓰는게 편함

```
#include <bits/stdc++.h>
using namespace std;

using ll = long long;
using PLL = pair<ll, ll>;

int N, K;
ll A[505005], D[505005], C[505005];

int T[1 << 17];
inline ll f(int i, int j){
    return D[i] + (A[j] - A[i]) * (j - i);
}

void update(int hi){
    int node = 1, s = 1, e = N;
    while(s <= e){
        int m = s + e >> 1;
        int &lo = T[node];
        if(f(hi, s) < f(lo, s)) swap(lo, hi);
        if(f(lo, e) < f(hi, e)) break;
        if(f(lo, m) <= f(hi, m)) node = node << 1 | 1, s = m + 1;
        if(f(hi, m) < f(lo, m)) swap(lo, hi), node = node << 1, e = m - 1;
    }
}

int query(int x){
    int node = 1, s = 1, e = N, line = 0;
    while(s <= e){
        int m = s + e >> 1;
        if(f(T[node], x) < f(line, x)) line = T[node];
        if(x < m) node = node << 1, e = m - 1;
        else if(x > m) node = node << 1 | 1, s = m + 1;
        else break;
    }
```

```
    return line;
}

PLL Solve(ll cost){
    memset(T, 0, sizeof T);
    for(int i=1; i<=N; i++){
        int prv = query(i);
        D[i] = f(prv, i) + cost;
        C[i] = C[prv] + 1;
        update(i);
    }
    return {D[N], C[N]};
}

int main(){
    ios_base::sync_with_stdio(false); cin.tie(nullptr);
    cin >> N >> K;
    for(int i=1; i<=N; i++) cin >> A[i];
    partial_sum(A+1, A+N+1, A+1);

    ll l = 0, r = 1e15;
    while(l < r){
        ll m = l + r >> 1;
        if(Solve(m).second <= K) r = m;
        else l = m + 1;
    }
    cout << Solve(r).first - r * K;
}
```